Chapter 18: NASIS Query Writing

This guide is designed to provide the NASIS user an understanding of SQL and its various uses in writing NASIS queries. SQL is used to write queries, reports, properties, calculations and validations. An understanding of the NASIS data structure (tables and columns) is required before using SQL to query or report data.

In addition to this document, the following found on the NASIS web site are references necessary for understanding this process:

- The "Table Structure Report" identifying the NASIS tables and the columns within each table.
- The "NASIS CVIR Language Manual" Scripting language for NASIS Calculations, Validations, Interpretations and Reports.
- Getting Started Chapter on Writing Custom Queries.
- NASIS database schemas.

SQL

To become adept at writing queries, the user must have knowledge of the Structured Query Language database language. SQL, as it is commonly referred to, was created by IBM in the early 1970s as a unified language for defining, querying, modifying and controlling the data in a relational database. There are now over 75 different flavors of SQL in commercial use. NASIS originally used the Informix database and is now using the Microsoft SQL Server database. The basic SQL structure is standardized between commercial databases however there are dialect differences. This document will focus on the SQL Server dialect and how it is used with the various soils databases. SQL is used in NASIS and the Soil Data Mart, the Soil Data Access site and Web Soil Survey. Understanding SQL will allow the user the ability to query data or write reports from these various databases and sites.

SQL Syntax

A SQL statement contains several elements. The SQL has certain "**Keywords**" that have special meaning. They are typically entered in UPPER CASE, however SQL is not case sensitive. This is done for organization purposes only. The statement also contains **identifiers** which are the names of the databases, tables and columns. Typically, identifiers are entirely in lower case. And, the statement contains **operators or functions** are used for comparisons or mathematical equations. The operator can be used for arithmetic (+ or **-**) or as comparisons (> or =) or as logical (AND, OR, NOT) or aggregate functions (MAX, MIN, SUM, COUNT, AVG).

Keywords

The basic SQL statement consists of 3 key words:

- **SELECT** (column)
- **FROM** (table)
- **WHERE** (condition)

The **SELECT** clause:

- specifies the columns (e.g. musym, muname, mukind) to be retrieved,
- each column must have a unique name,
- allows for expressions that must follow normal SQL syntax (e.g. sandtotal_r + silttotal_r + claytotal_r AS particle_size),

- if expressions are used in the select statement, an alias (e.g. “particle_size”) must be used with the expression to provide a unique name.

The **FROM** clause:

- specifies all the tables used in the query,
- and may specify aliases and outer joins.

The **WHERE** clause:

- filters which rows to use in the FROM clause
- uses normal SQL conditions and
- uses the NASIS "JOIN table TO table" syntax to simplify writing join conditions,
- and the two tables in a JOIN condition must have a relationship

Example:

```
SELECT musym, muname
FROM mapunit
WHERE muname = ‘Harney silt loam, 0 to 1 percent slopes’
```

Identifiers

The NASIS and SSURGO metadata reports, found on the appropriate web sites, contain the identifiers needed for SQL statements. The “Tables_and_Columns.pdf” document provides the list of tables and the columns within each. These documents are designed to provide the user with information necessary to write SQL statements.

Operators or Functions

The arithmetic operators, comparison operators, logical operators and the aggregate functions are used to filter the search functions of the WHERE clause. In this document on page 6 a chart is provided that identifies the data types and the various comparison operators that can be used. Further information on the various operators and functions will be discussed as they are introduced in this document.

NASIS SQL

Queries

Basic Queries

The purpose of a NASIS query is to load the selected set with data that is filtered to meet the needs of the user. The NASIS "query" requires knowledge of SQL and database structure. The NASIS query has two basic parts: the **FROM** clause and the **WHERE** clause. The **SELECT** clause is not used since a NASIS query is designed to return the data for all the columns within the table(s) identified in the FROM clause. Since all queries are understood to pull all columns, the SELECT * (select all columns) is understood and written into the Query editor.

The first step is to identify the data that is to be loaded into the selected set.

Load all instances of a named component

Assuming a simple query, to load all instances of a particular component name, then the next step in writing the query is to review the "Tables and Columns" report:

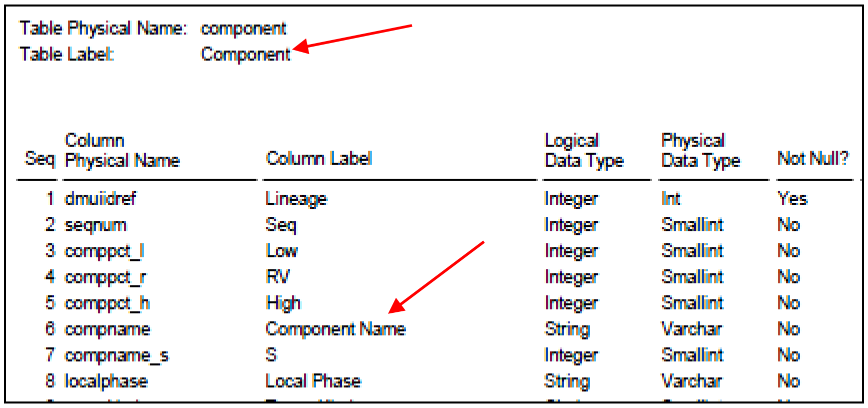
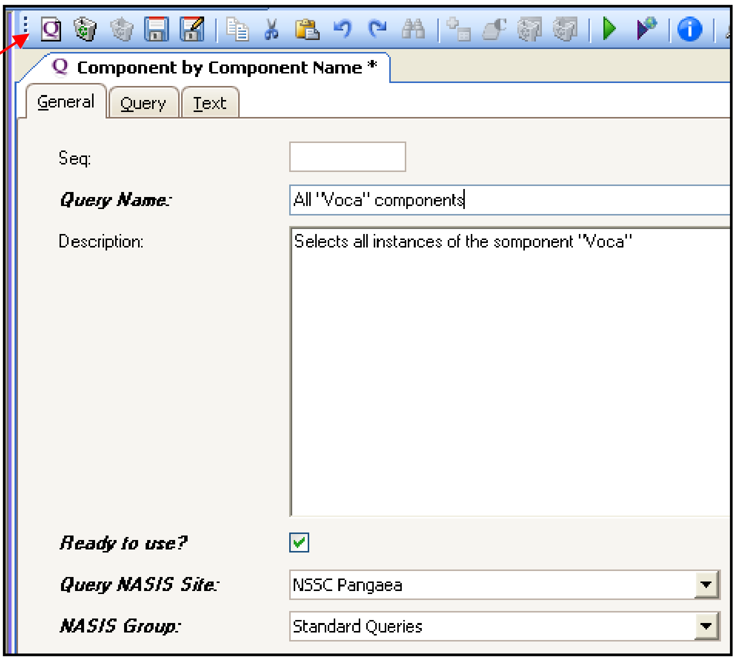


Table Physical Name: component
Table Label: Component

Seq	Column Physical Name	Column Label	Logical Data Type	Physical Data Type	Not Null?
1	dmuiidref	Lineage	Integer	Int	Yes
2	seqnum	Seq	Integer	Smallint	No
3	comppct_l	Low	Integer	Smallint	No
4	comppct_r	RV	Integer	Smallint	No
5	comppct_h	High	Integer	Smallint	No
6	compname	Component Name	String	Varchar	No
7	compname_s	S	Integer	Smallint	No
8	localphase	Local Phase	String	Varchar	No

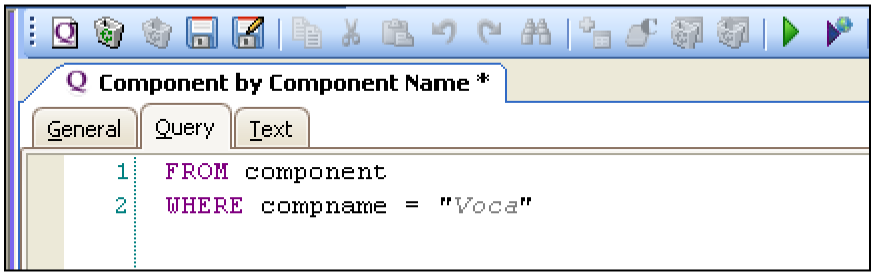
The component name column (Physical Name is "compname") is found in the Component table and the field is a variable character (Varchar).

In NASIS, click on the "Add New Query" icon . The General tab appears. Populate the query name and the description. Both fields are required.



Seq:
Query Name: All "Voca" components
Description: Selects all instances of the somponent "Voca"
Ready to use? ✓
Query NASIS Site: NSSC Pangaea
NASIS Group: Standard Queries

The Query tab is used to write the SQL. The "SELECT *" is understood for all queries, therefore the query begins with the keyword **FROM**.



```
FROM component
WHERE compname = "Voca"
```

The above query is an example of a basic query. It includes one table in the FROM clause and one condition in the WHERE clause. This example will return and populate all component data for all instances where the component name is equal to the exact letters "Voca".

The equal sign "=" is a "comparison operator". The equal sign is used for an exact match in the field being compared. Any component in which the name is "VOCA" or "voca" or "VoCa" will not be loaded.

This query can be run against the national database in order to populate the local database, or against the local database to populate the selected set.

Load all map units for a given map unit name
Use of the Question mark "?"

The user wishes to design a query to load by map unit name, however there are multiple map unit names the user wishes to use. The user is interested in writing one query that can be used multiple times.

The first step is to identify the Table and Columns:

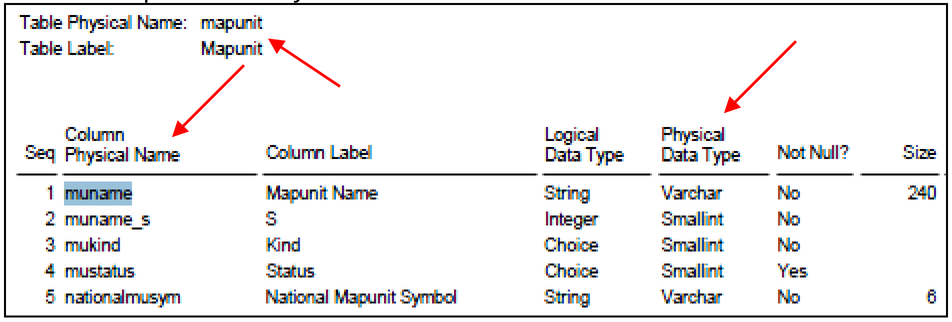


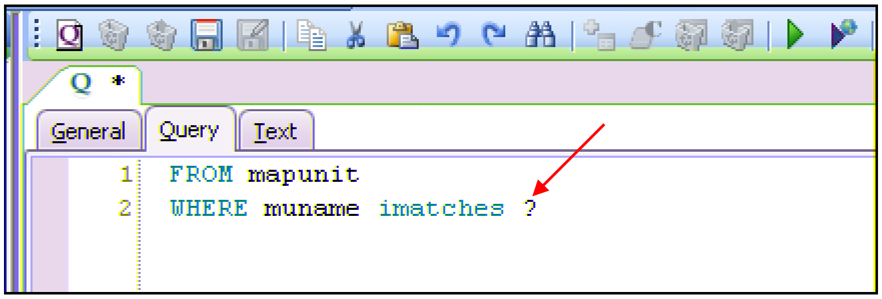
Table Physical Name: mapunit
Table Label: Mapunit

Seq	Column Physical Name	Column Label	Logical Data Type	Physical Data Type	Not Null?	Size
1	muname	Mapunit Name	String	Varchar	No	240
2	muname_s	S	Integer	Smallint	No	
3	mukind	Kind	Choice	Smallint	No	
4	mustatus	Status	Choice	Smallint	Yes	
5	nationalmusym	National Mapunit Symbol	String	Varchar	No	6

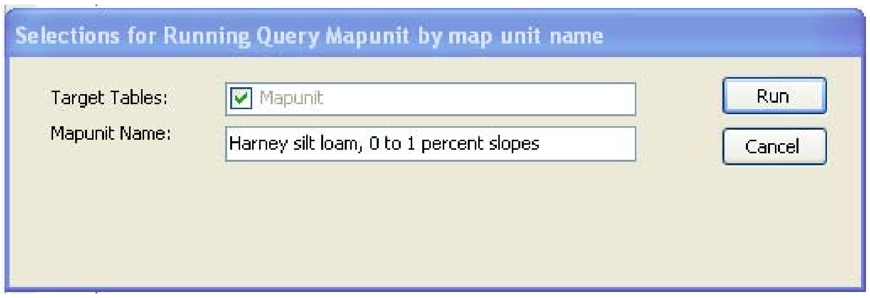
The map unit name is in the map unit table. The map unit name column is identified with the physical name of "muname". The "muname" column is a Variable Character.

This query is designed with a variable (the question mark ?) allowing the user to write one query for multiple needs. The **WHERE** clause in this query uses the comparison operator "imatches" (case isinsensitive) and uses a question mark (?).

The "?" question mark creates a parameter box. The parameter box is used to identify the map unit name to be queried. When run, the query will prompt the user to identify the map unit.



```
FROM mapunit
WHERE muname imatches ?
```



Selections for Running Query Mapunit by map unit name

Target Tables:	✓ Mapunit
Mapunit Name:	Harney silt loam, 0 to 1 percent slopes

Load all components by name and specific component percentage
Use of ">=" comparison operator

Once again, the first step is to use the Tables and Columns report to identify the columns for use in the query:

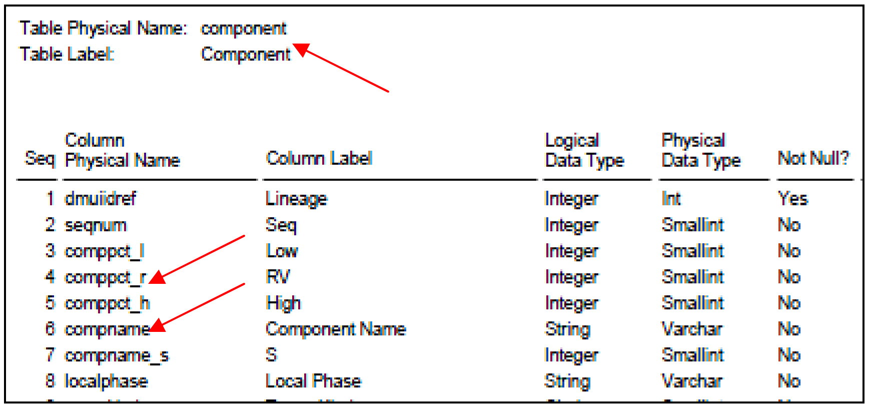
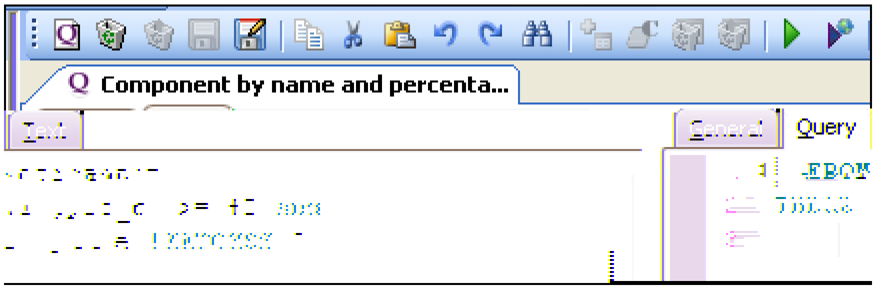


Table Physical Name: component
Table Label: Component

Seq	Column Physical Name	Column Label	Logical Data Type	Physical Data Type	Not Null?
1	dmuiidref	Lineage	Integer	Int	Yes
2	seqnum	Seq	Integer	Smallint	No
3	comppct_l	Low	Integer	Smallint	No
4	comppct_r	RV	Integer	Smallint	No
5	comppct_h	High	Integer	Smallint	No
6	compname	Component Name	String	Varchar	No
7	compname_s	S	Integer	Smallint	No
8	localphase	Local Phase	String	Varchar	No

Notice the component percentage has three columns; one for the Low, the RV and the High. The user chooses the RV value since it is most commonly populated.



```
[illegible]
[illegible] >= [illegible]
[illegible]
```

The comparison operator “greater than or equal to” (>=) is used to search for those component names in which the component percentage is greater than or equal to 80 percent.

Load all components by name, major component flag and component kind
Use of “?” and “(?)” parameter

The first step is to identify the Tables and columns to be used. The component table contains all three search conditions.
The component name is a “Variable Character”.
The Component Kind is a “Choice” field.
The major component flag is a “Boolean” (or commonly a Yes or No field).

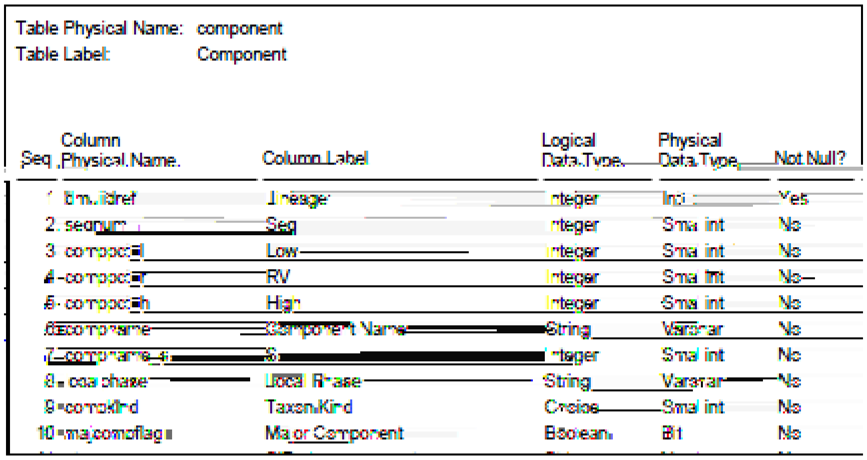
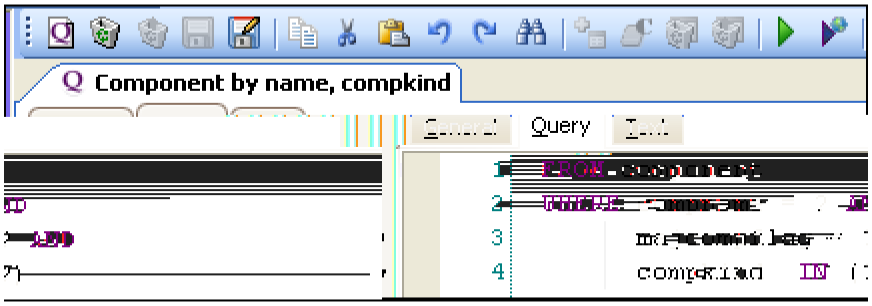


Table Physical Name: component
Table Label: Component

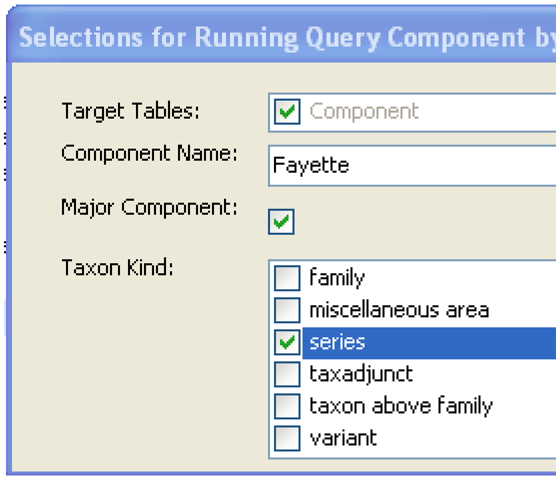
Seq	Column Physical Name	Column Label	Logical Data Type	Physical Data Type	Not Null?
1	dmuiidref	Lineage	Integer	Int	Yes
2	seqnum	Seq	Integer	Small int	No
3	comppct_l	Low	Integer	Small int	No
4	comppct_r	RV	Integer	Small int	No
5	comppct_h	High	Integer	Small int	No
6	compname	Component Name	String	Varchar	No
7	compname_s	S	Integer	Small int	No
8	localphase	Local Phase	String	Varchar	No
9	compkind	Taxon Kind	Choice	Small int	No
10	majcompflag	Major Component	Boolean	Bit	No

Opening NASIS and write the query. The “Choice” field for component kind allows the use of a new search condition – the question mark enclosed in parentheses (?). This provides the field choice list as part of the parameter box.



```
FROM component
WHERE compname ...
[illegible]
compkind IN (?
```

Notice the parameter box now includes the choices for component kind

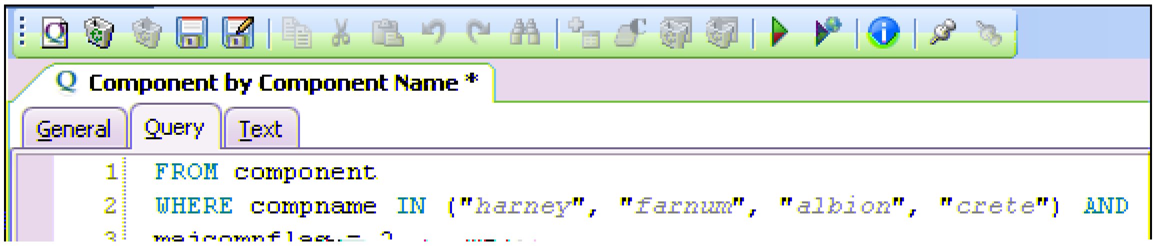


Selections for Running Query Component by

Target Tables:	✓ Component
Component Name:	Fayette
Major Component:	✓
Taxon Kind:	☐ family ☐ miscellaneous area ✓ series ☐ taxadjunct ☐ taxon above family ☐ variant

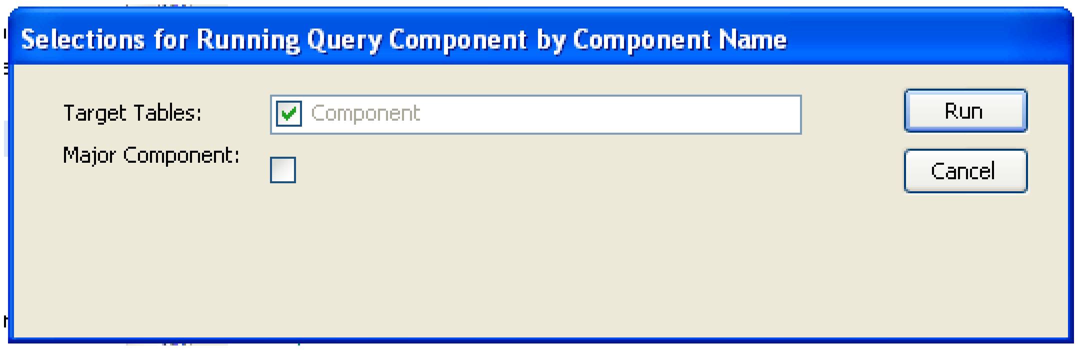
Load multiple components by name, major component flag
Use of "IN ( )" parameter

The "IN" command can be used to provide a list of variables to be searched. In this query, the user is loading several component names using a single query. The syntax used is the IN followed by a space, then open parentheses, then open quotation followed by the variable followed by the closed quote. Variables are strung using a comma to separate each variable. The last variable is followed with a closed parenthesis.



```
FROM component
WHERE compname IN ("harney", "farnum", "albion", "crete") AND
```

This query will prompt for "Major Component". If checked, then only major components of the listed soils will be loaded. If left unchecked, all components with these names will be loaded.



Target Tables: ☑ Component
Major Component: ☐

Load components by name and specific slope range
Use of BETWEEN command

The user would like a query that identifies a certain component name and the user wishes to supply a slope range to load components.

Reviewing the Tables and Columns report, the component table and its columns are identified.

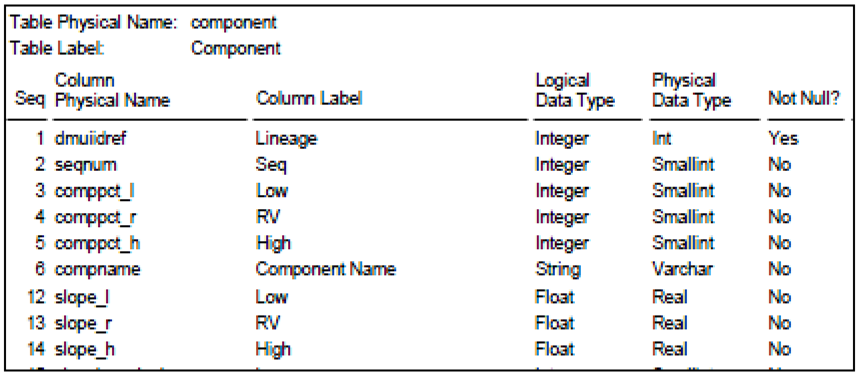
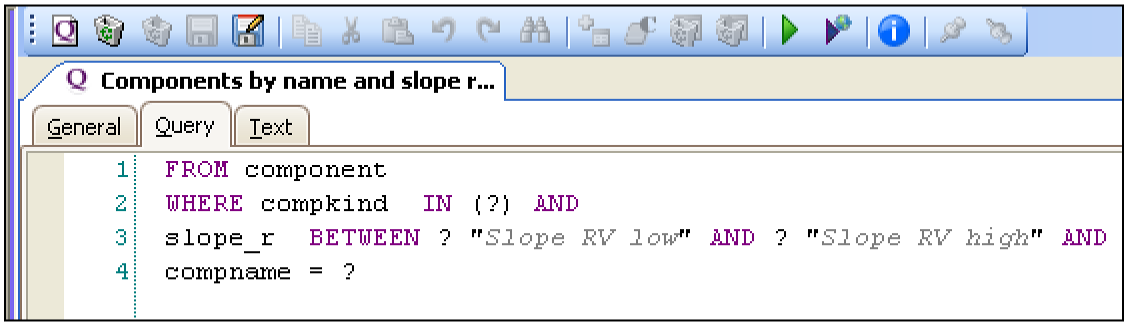


Table Physical Name: component
Table Label: Component

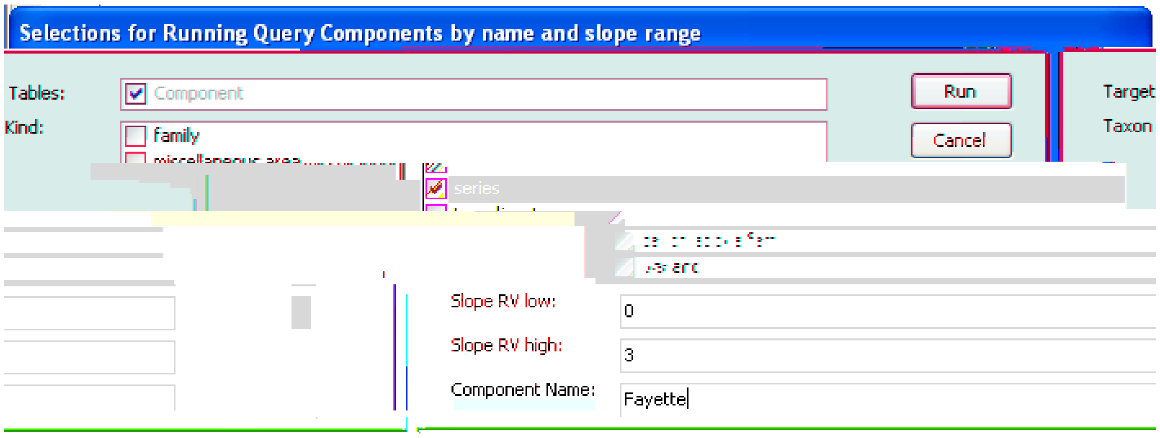
Seq	Column Physical Name	Column Label	Logical Data Type	Physical Data Type	Not Null?
1	dmuiidref	Lineage	Integer	Int	Yes
2	seqnum	Seq	Integer	Smallint	No
3	comppct_l	Low	Integer	Smallint	No
4	comppct_r	RV	Integer	Smallint	No
5	comppct_h	High	Integer	Smallint	No
6	compname	Component Name	String	Varchar	No
12	slope_l	Low	Float	Real	No
13	slope_r	RV	Float	Real	No
14	slope_h	High	Float	Real	No

The slope is an integer with the data type of Real. The parameters will be specific numbers. The use of the BETWEEN command allows the user to prompt for two fields to be entered in the parameter box.



```
FROM component
WHERE compkind  IN (?) AND
slope_r  BETWEEN ? "Slope RV low" AND ? "Slope RV high" AND
compname = ?
```

The query prompts the user for two fields to identify the user slope range. In addition, notice the use of the words in quotes after each question mark. Placing the words “Slope RV low” after the question mark allows the user to override the default label and assign a more meaningful label for the specific field.


Selections for Running Query Components by name and slope range
Run
Cancel
Target
Taxon

Tables: ✓ Component
Kind: family
series ✓

Slope RV low: 0
Slope RV high: 3
Component Name: Fayette

Data types and comparison operators

Understanding data types (integer, character, etc.) is important when writing queries because comparison operators used in query conditions are valid with some data types but not with others. Consequently, when a query is written to specify a condition in the WHERE clause, there must be a comparison operator (such as = or MATCHES) that is compatible with the data element in the query conditions. For example, the data element "area name" is a "Variable Character" data type and the MATCHES operator is valid for this data type. (MATCHES is case sensitive, IMATCHES is case insensitive, = is exact match)

Data Types| **Comparison Operators**

Notes: Date and date time values must be entered in the correct format or an SQL error will result. NOT, AND, and OR operators are used to combine two conditions; they are not related to data type.

Blank	Allowed
II	Allowed by query program, but results may not be meaningful
III	Allowed by query program, but will result in SQL error when query is executed.
IV	Not allowed

Exercises

Using the references, write the following queries.

1. Load all map units with a given name with a status of “correlated”.
2. Develop a query that can be used to identify a map unit name with a specific Farmland Class.
3. Load a component based on its drainage class.
4. Load a map unit by name that ranges from 10,000 to 20,000 acres.
5. Load all components that are mollic albaqualfs.

Advanced Queries

In most instances the user will wish to load several tables of data into the local database or selected set. Additional filters (search conditions) may also be necessary to target certain data sets for editing purposes.

Adding additional tables
Loading specific components with specific horizon depths

Review the "Tables and Columns" report to identify the horizon table and its columns.

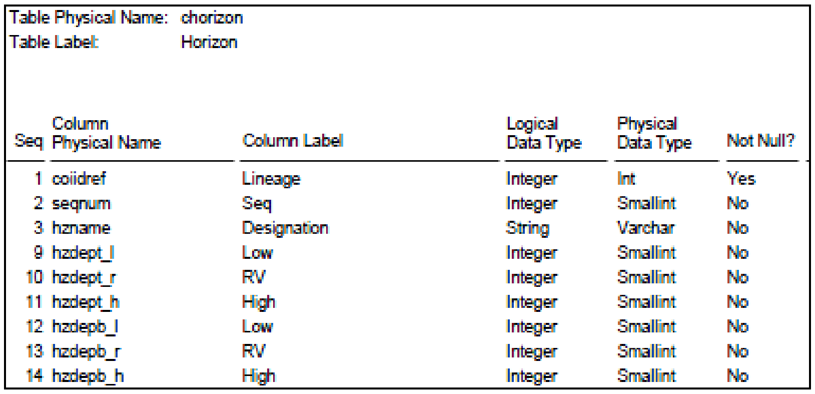
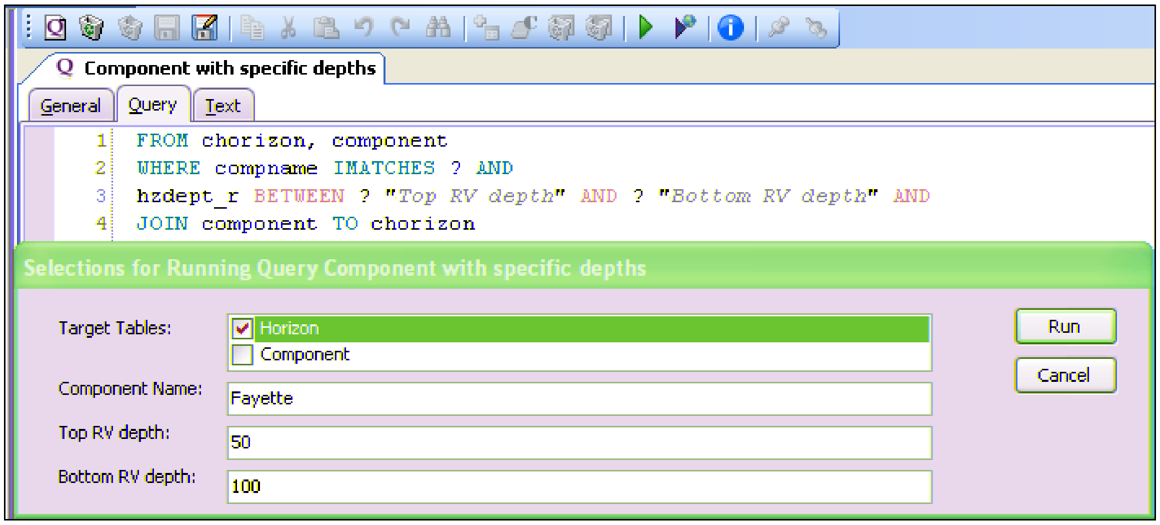


Table Physical Name: chorizon
Table Label: Horizon

Seq	Column Physical Name	Column Label	Logical Data Type	Physical Data Type	Not Null?
1	coiidref	Lineage	Integer	Int	Yes
2	seqnum	Seq	Integer	Smallint	No
3	hzname	Designation	String	Varchar	No
9	hzdept_l	Low	Integer	Smallint	No
10	hzdept_r	RV	Integer	Smallint	No
11	hzdept_h	High	Integer	Smallint	No
12	hzdepb_l	Low	Integer	Smallint	No
13	hzdepb_r	RV	Integer	Smallint	No
14	hzdepb_h	High	Integer	Smallint	No

Item 10, "hzdept_r" is the top depth Representative value. This will be used to identify those horizons that fall within the users determined limits.

The query

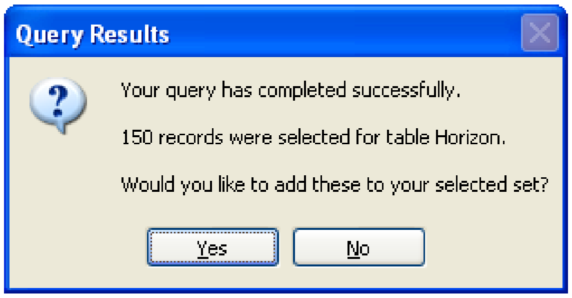

Component with specific depths
General
Query
Text
Selections for Running Query Component with specific depths
Run
Cancel

```
FROM chorizon, component
WHERE compname IMATCHES ? AND
hzdept_r BETWEEN ? "Top RV depth" AND ? "Bottom RV depth" AND
JOIN component TO chorizon
```

Target Tables: [✓ Horizon] Component
Component Name: Fayette
Top RV depth: 50
Bottom RV depth: 100

includes two tables in the FROM clause separated by a comma. The clause "WHERE" includes the search conditions for the component name and uses the previously defined BETWEEN to establish a range of top depths for the search. The query is completed with the JOIN statement joining the two tables, component and chorizon.

The Target Table is set to Horizon because horizon depth is the focus of the query. The user then populates the component name to search and the top and bottom limits of the horizon top depth.



Query Results

Your query has completed successfully.

150 records were selected for table Horizon.

Would you like to add these to your selected set?

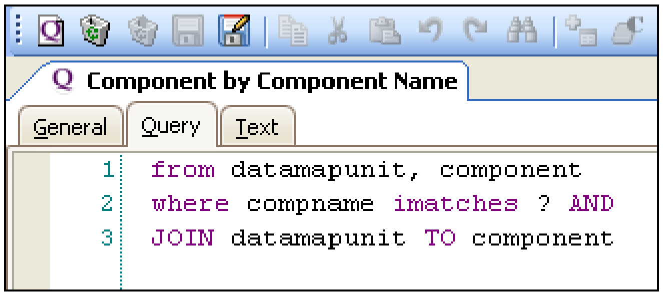
Target Tables

Simply put, the target table focuses the outcome of a particular query. In this way, the user can control the query so that it loads only the specific data to be worked on during that edit session. The target table can greatly restrict or expand the number of records returned by a particular query. To understand target tables, the user must understand the relationship between objects in the NASIS database. The data model diagrams help to visualize this relationship.

So, how does a Target Table restrict the records returned by a query? In an edit session, the user only wishes to work with components that are named “Farnum”. The user would choose a query that loads components by ***compname*** and specify **Farnum**. Because component name is in the component table, either datamapunit or component could be selected as the target table. *Whether or not only the* ***Farnum series*** *is loaded depends on the target table choice*.

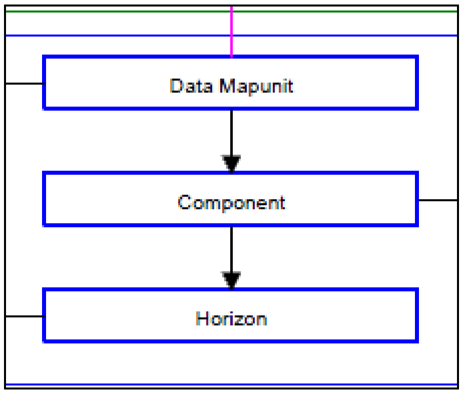
- If *datamapunit* is selected as the target table, *all* Data Mapunits that have at least one Farnum component, in addition to *all* the DMU other components, are loaded into the component table.
- If *component* is selected as the target table, only components named Farnum are loaded.

Using this simple query as an example:

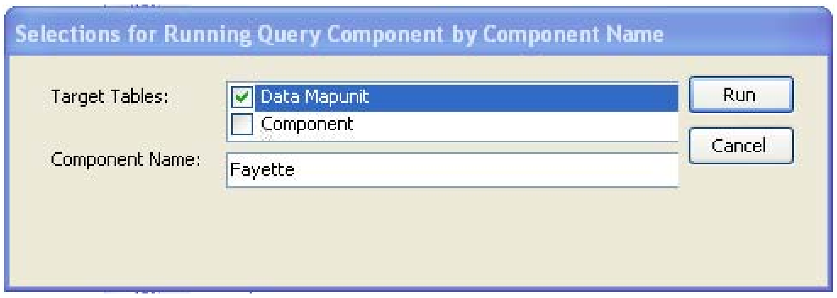


Component by Component Name

```
from datamapunit, component
where compname imatches ? AND
JOIN datamapunit TO component
```



The query has two tables in the FROM clause that become Target Tables in the parameter box. Setting the Target Table to “Data Mapunit”

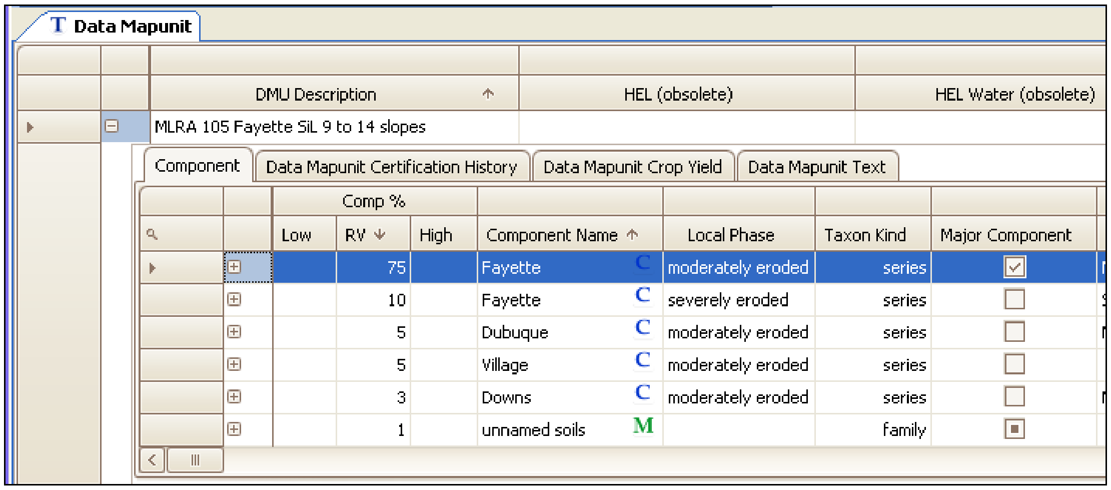


Selections for Running Query Component by Component Name

Target Tables: [✓ Data Mapunit] Component

Component Name: Fayette

Provides for a selected set in which all Data Mapunits in which the Fayette component is a member.



Data Mapunit

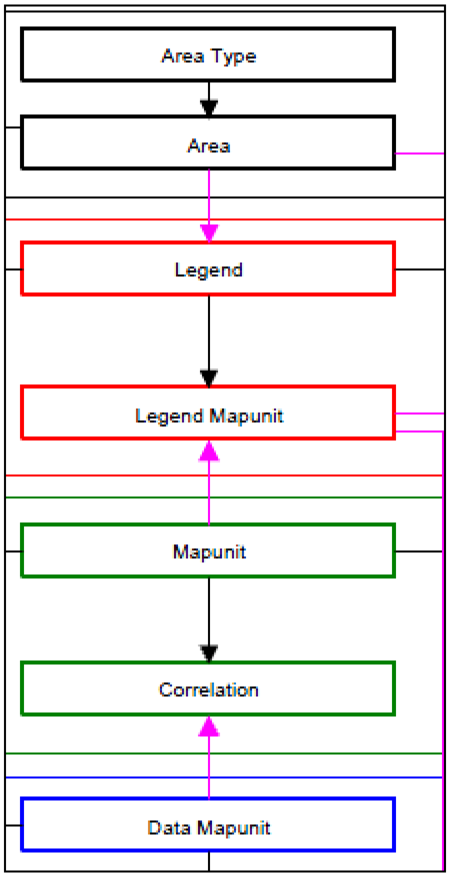
DMU Description	HEL (obsolete)	HEL Water (obsolete)
MLRA 105 Fayette SiL 9 to 14 slopes		

Component | Data Mapunit Certification History | Data Mapunit Crop Yield | Data Mapunit Text

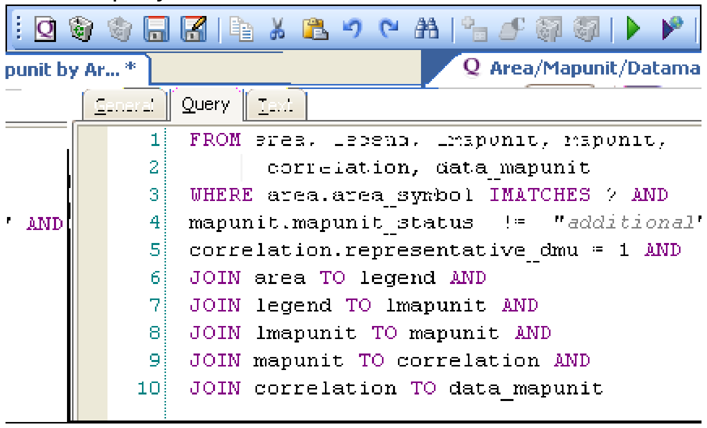
Comp %						
Low	RV	High	Component Name	Local Phase	Taxon Kind	Major Component
	75		Fayette C	moderately eroded	series	✓
	10		Fayette C	severely eroded	series	☐
	5		Dubuque C	moderately eroded	series	☐
	5		Village C	moderately eroded	series	☐
	3		Downs C	moderately eroded	series	☐
	1		unnamed soils M		family	■

In this example the user will load all map units and the associated data mapunit for a given Survey Area. The manuscript reports require the Area, Legend, Mapunit, and Datamapunit objects to be loaded. This query will require multiple tables and additional search conditions.

Using the database schema, the user identifies the tables needed for this query:



After the tables are identified the user can review the Tables and Columns report and enter the appropriate information into the NASIS query:



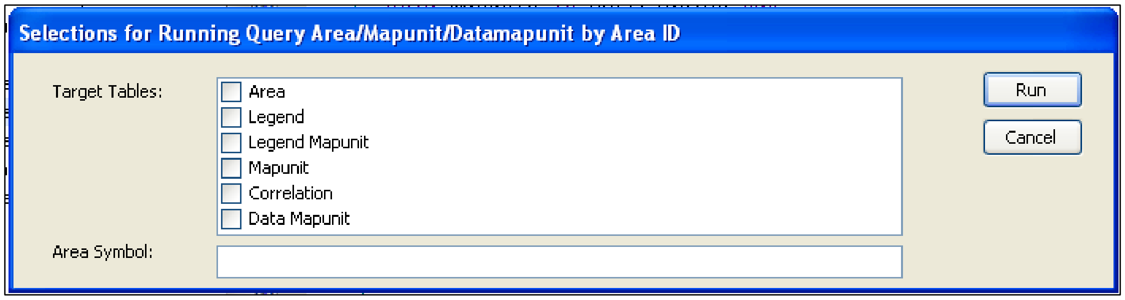
```
 1  FROM area, legend, lmapunit, mapunit,
 2          correlation, data_mapunit
 3  WHERE area.area_symbol IMATCHES ? AND
 4  mapunit.mapunit_status  !=  "additional'
 5  correlation.representative_dmu = 1 AND
 6  JOIN area TO legend AND
 7  JOIN legend TO lmapunit AND
 8  JOIN lmapunit TO mapunit AND
 9  JOIN mapunit TO correlation AND
10  JOIN correlation TO data_mapunit
```

The required tables are entered in the FROM clause. The clause "WHERE" has three search conditions (lines 3 through 5) and five JOIN conditions (lines 6 through 10).

The search conditions include the area symbol (using an IMATCHES), the map unit status (using "!=" does not equal comparison operator) and the representative datamapunit (using equal comparison operator) as assigned in the correlation table.

The JOIN conditions (lines 6 through 10) are used to join the various tables as assigned in the schema. The JOIN condition syntax is written "JOIN {table} TO {table} AND". The logical operator "AND" is used to join all tables within the JOIN clause.

This query prompts for the Target Tables and Area Symbol.

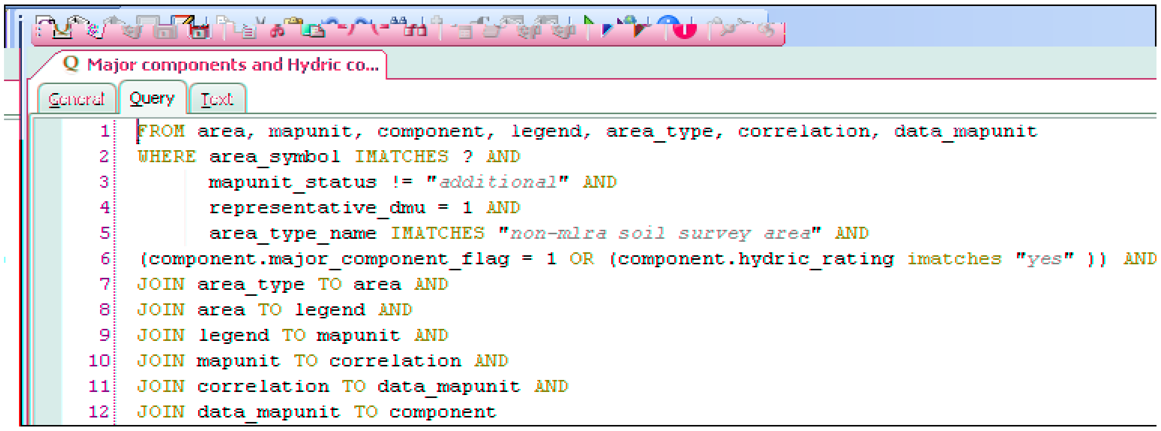


Selections for Running Query Area/Mapunit/Datamapunit by Area ID

Target Tables:
- [ ] Area
- [ ] Legend
- [ ] Legend Mapunit
- [ ] Mapunit
- [ ] Correlation
- [ ] Data Mapunit

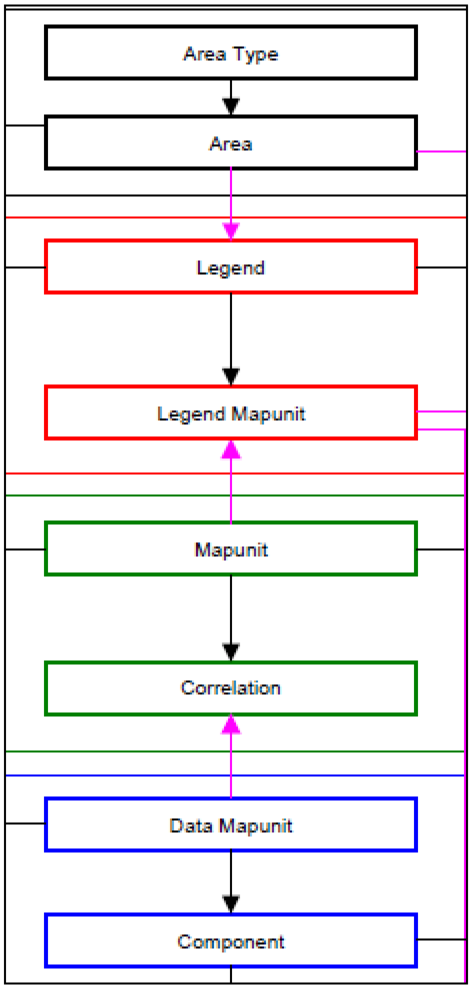
Area Symbol:

Use of the OR command: Load all major components and all hydric components



```
 1  FROM area, mapunit, component, legend, area_type, correlation, data_mapunit
 2  WHERE area_symbol IMATCHES ? AND
 3         mapunit_status != "additional" AND
 4         representative_dmu = 1 AND
 5         area_type_name IMATCHES "non-mlra soil survey area" AND
 6  (component.major_component_flag = 1 OR (component.hydric_rating imatches "yes" )) AND
 7  JOIN area_type TO area AND
 8  JOIN area TO legend AND
 9  JOIN legend TO mapunit AND
10  JOIN mapunit TO correlation AND
11  JOIN correlation TO data_mapunit AND
12  JOIN data_mapunit TO component
```

Explanation



This is a multi-table query that prompts the user to enter a
Survey Symbol.

Line 1: Include all the tables from Area to Component. Notice
the list of tables is not in logical order, but in the
order to assign Target Tables. The Area,
Mapunit and Component are the Target Tables.
This query is used to load components,
therefore the component table is required since
the major component flag and the hydric rating are in
the component table and therefore this table is
required as a Target Table.

Line 2: The question mark "?" creates a parameter box.
The IMATCHES allows the user to enter data
insensitive to case; the match will be made on the
letters not the case.

Line 3: If "Mapunit" is chosen as the Target Table, then
this query will not load map units with the status
labeled as "additional".

Line 4: If there are multiple records in the correlation table,
then only the one record identified as the
representative DMU is loaded.

Line 5: Only the legends with the Area Type "Non- MLRA
Soil Survey Area" will be loaded.

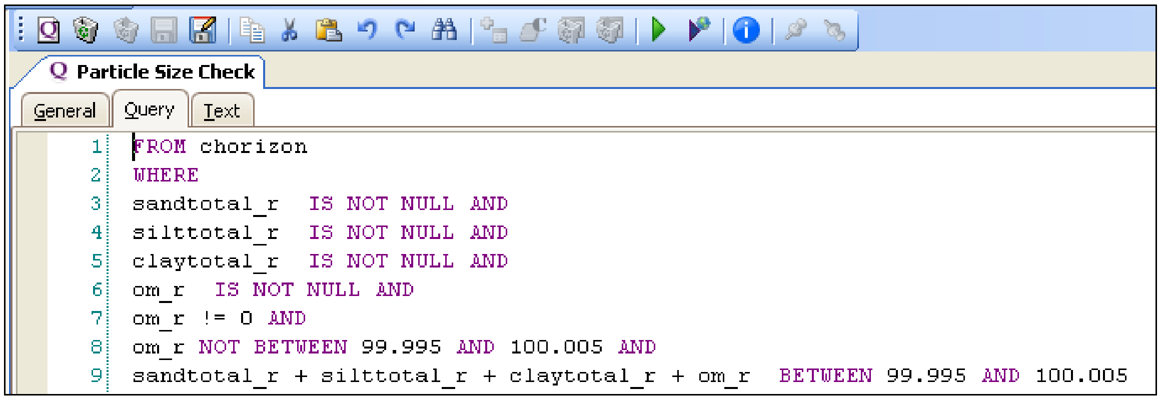
Use of the OR command:

Line 6: Notice there are two search conditions within a set of
parentheses. The two search conditions are
compared using the "OR" command. This search
condition states "load all the major
components OR load all the components in which
the hydric rating matches "Yes". By using the OR
command in this search condition, the computer will load all the major components and any
hydric components whether the hydric component is a major or minor component.

Lines 7 – 12: The JOIN conditions joining all the tables in the logical sequence found on the database
schema.

Arithmetic Operators
Load horizons in which Sand, Silt and Clay totals do not equal 100

After checking the Tables and Columns report, the sand, silt and clay columns are entered into the query.



```
1  FROM chorizon
2  WHERE
3  sandtotal_r  IS NOT NULL AND
4  silttotal_r  IS NOT NULL AND
5  claytotal_r  IS NOT NULL AND
6  om_r  IS NOT NULL AND
7  om_r != 0 AND
8  om_r NOT BETWEEN 99.995 AND 100.005 AND
9  sandtotal_r + silttotal_r + claytotal_r + om_r  BETWEEN 99.995 AND 100.005
```

Line 1:	selects the horizon table. The physical name is “chorizon”.
Line 2:	begins the WHERE clause
Lines 3-5:	verify that the sand, silt and clay fields for this query can not be NULL fields.
Lines 6-8:	checks that the representative value for OM can not be NULL or Zero and must not be 100.
Line 9:	adds the sand, silt, clay and OM RVs to verify they are between the two values.

If these conditions are met, then the query will load those horizons that meet these search criteria. Notice, there are no prompts for a parameter box. This query will load all horizons in the national database that meet these conditions.

Exercise:

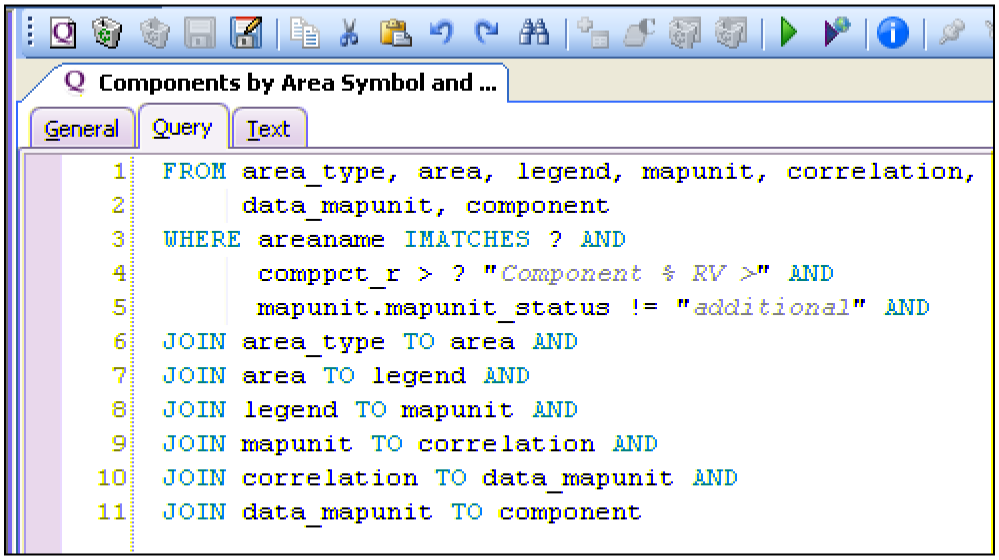
How can this query be tailored to limit the size of the returned rows?

Outer Joins

Load all Components for a survey area based on component percent

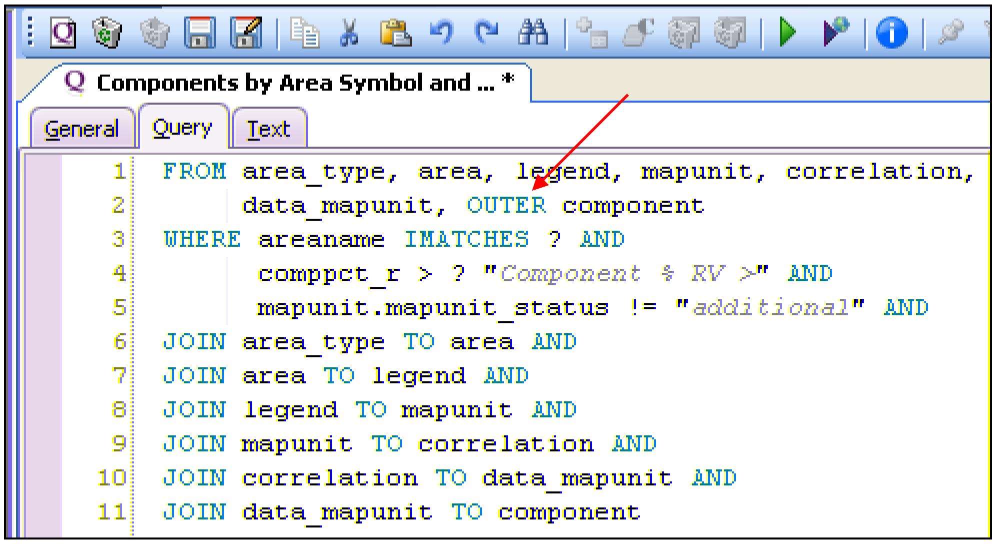
This following query was used to explain Target Tables. It will be used again to explain OUTER Joins.

When using multiple tables, the query assumes a one to one relationship between the two tables. There is a row in the parent table (Data Mapunit) that is linked directly to the child table (Component). If a data mapunit has an empty component table then the 1 to 1 match fails and that data mapunit does not meet the criteria.



```
FROM area_type, area, legend, mapunit, correlation,
     data_mapunit, component
WHERE areaname IMATCHES ? AND
       comppct_r > ? "Component % RV >" AND
       mapunit.mapunit_status != "additional" AND
JOIN area_type TO area AND
JOIN area TO legend AND
JOIN legend TO mapunit AND
JOIN mapunit TO correlation AND
JOIN correlation TO data_mapunit AND
JOIN data_mapunit TO component
```

So, what if a Data Mapunit has a Component table in which no data exists? Should that information be loaded as part of the selected set? Most times the answer is Yes. This is over come by the use of the OUTER join.



```
FROM area_type, area, legend, mapunit, correlation,
     data_mapunit, OUTER component
WHERE areaname IMATCHES ? AND
       comppct_r > ? "Component % RV >" AND
       mapunit.mapunit_status != "additional" AND
JOIN area_type TO area AND
JOIN area TO legend AND
JOIN legend TO mapunit AND
JOIN mapunit TO correlation AND
JOIN correlation TO data_mapunit AND
JOIN data_mapunit TO component
```

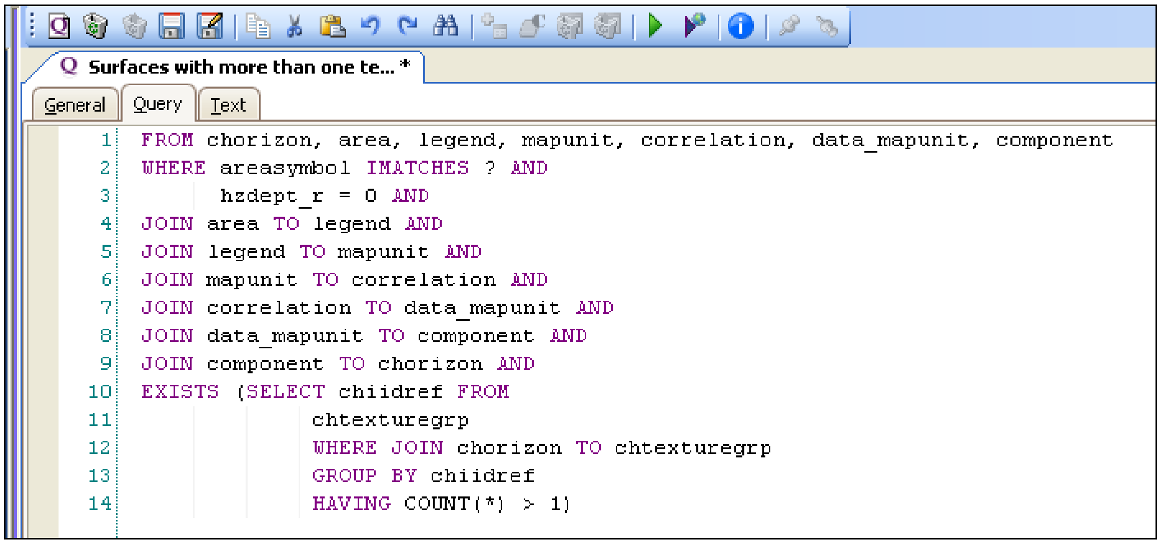
With the query modified, it will now load all Data Mapunits associated with the map unit even if there is nothing populated in the Component table. The OUTER join stipulates that all data mapunits are to be loaded whether or not there is data in the component table.

Subqueries

A subquery is used to further restrict the results of the main query. A subquery is set within the query using parentheses.

EXISTS

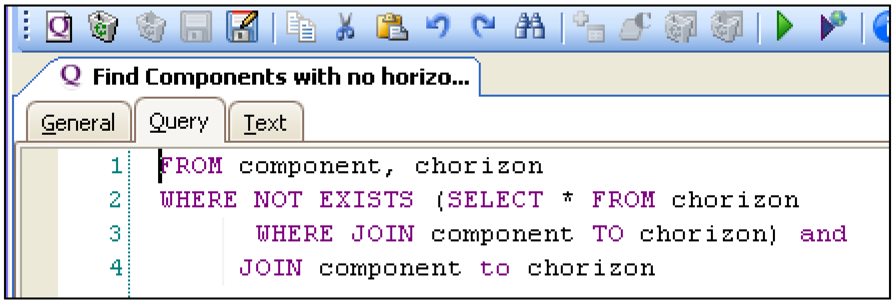
The EXISTS operator is testing for existence of data, therefore only one column or the asterisk (*) is necessary is in the SELECT statement. What if a query is written to load all components that have more than one texture in the surface horizon? In this example, the query is written to prompts for a survey area and to select the surface horizon. The subquery begins with EXISTS and tests the existence of more than one record ID (chiidref) in the horizon texture group table for the surface layer. Notice the use of the chorizon table which links the subquery to the query.



```
FROM chorizon, area, legend, mapunit, correlation, data_mapunit, component
WHERE areasymbol IMATCHES ? AND
      hzdept_r = 0 AND
JOIN area TO legend AND
JOIN legend TO mapunit AND
JOIN mapunit TO correlation AND
JOIN correlation TO data_mapunit AND
JOIN data_mapunit TO component AND
JOIN component TO chorizon AND
EXISTS (SELECT chiidref FROM
              chtexturegrp
              WHERE JOIN chorizon TO chtexturegrp
              GROUP BY chiidref
              HAVING COUNT(*) > 1)
```

NOT EXISTS

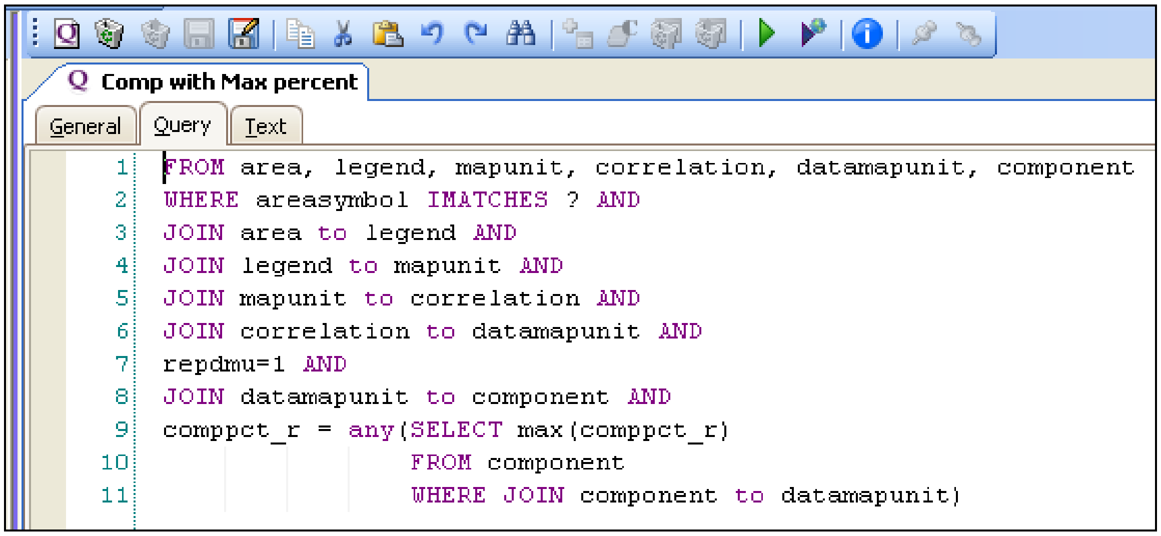
Contrary to EXISTS, the NOT EXISTS is identifying the non existence of data. What if it was necessary to identify those components in which no horizon information is entered? The outer join is helpful, however another method is available. Using a subquery can be helpful to identify a child table with no open rows. In this example the NOT EXISTS is used. The subquery is in parentheses and it states to select everything for the horizon table and joins the component and the horizon table. The NOT EXISTS is a negative or reversal. If nothing exists, a table with no data, then it returns a TRUE statement and that data is loaded into the selected set.



```
FROM component, chorizon
WHERE NOT EXISTS (SELECT * FROM chorizon
    WHERE JOIN component TO chorizon) and
   JOIN component to chorizon
```

ANY operator

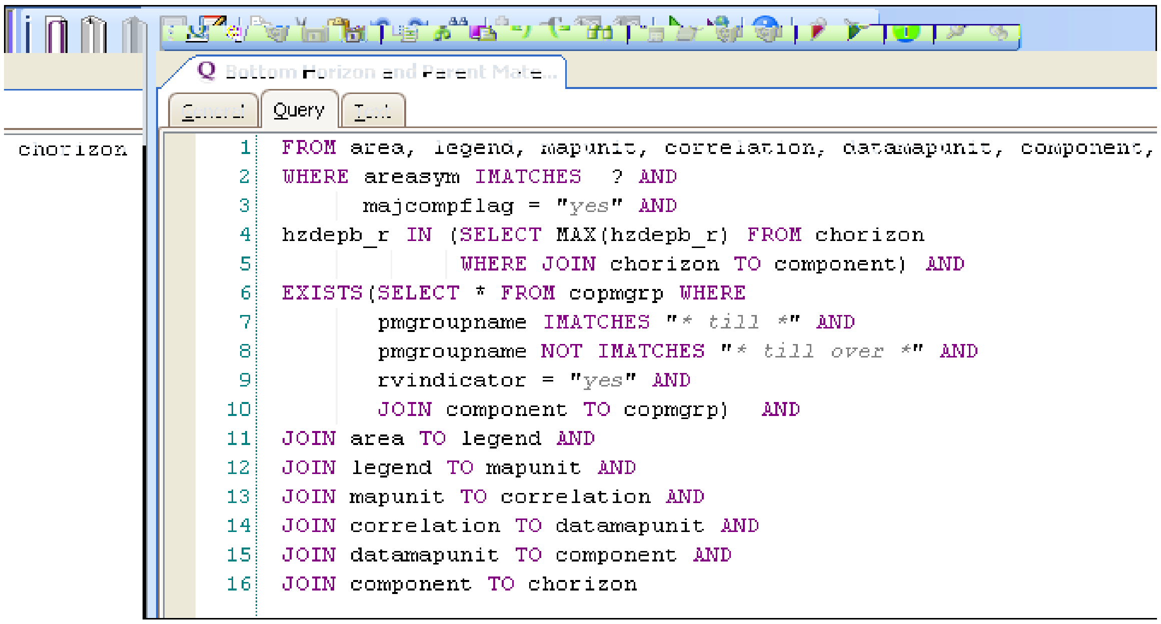
What if it was necessary to identify the component with the maximum percentage in data mapunit? One method is through a subquery using the ANY operator. Notice the query prompts for the area symbol, includes all the tables to component and has all the joins. In addition, a subquery is set apart using parentheses. The subquery will find the max component percentage and set that value to comppct_r. In doing so, the query will load the component that contains that maximum percentage.



```
FROM area, legend, mapunit, correlation, datamapunit, component
WHERE areasymbol IMATCHES ? AND
JOIN area to legend AND
JOIN legend to mapunit AND
JOIN mapunit to correlation AND
JOIN correlation to datamapunit AND
repdmu=1 AND
JOIN datamapunit to component AND
comppct_r = any(SELECT max(comppct_r)
               FROM component
               WHERE JOIN component to datamapunit)
```

IN operator

The IN operator requires that the subquery returns one value. In this example the IN operator is identifying the maximum bottom depth of the soil – the use of the term MAX on the horizon bottom depth column. In addition, this query contains a second subquery to identify those soils in which the parent material is till or “till over”. This is an example of using multiple subqueries to load data.



```
FROM area, legend, mapunit, correlation, datamapunit, component,
WHERE areasym IMATCHES  ? AND
      majcompflag = "yes" AND
hzdepb_r IN (SELECT MAX(hzdepb_r) FROM chorizon
             WHERE JOIN chorizon TO component) AND
EXISTS(SELECT * FROM copmgrp WHERE
       pmgroupname IMATCHES "* till *" AND
       pmgroupname NOT IMATCHES "* till over *" AND
       rvindicator = "yes" AND
       JOIN component TO copmgrp)  AND
JOIN area TO legend AND
JOIN legend TO mapunit AND
JOIN mapunit TO correlation AND
JOIN correlation TO datamapunit AND
JOIN datamapunit TO component AND
JOIN component TO chorizon
```